

파이썬 프로그래밍 포트폴리오



파이썬 프로그래밍
컴퓨터정보공학과 20173851 황서희

목 차



Chapter 1 : 파이썬 언어의 개요와 첫 프로그래밍

Chapter 2 : 파이썬 프로그래밍을 위한 기초 다지기

Chapter 3 : 일상에서 활용되는 문자열과 논리 연산

Chapter 4 : 일상생활과 비유되는 조건과 반복

Chapter 5 : 항목의 나열인 리스트와 튜플

Chapter 6 : 키와 값의 나열인 딕셔너리와 중복을 불허하는 집합

소감

Chapter 1

파이썬 언어의 개요와 첫 프로그래밍



Section 01 파이썬 언어와 컴퓨팅 사고력

Section 02 파이썬 설치와 파이썬 쉘 실행

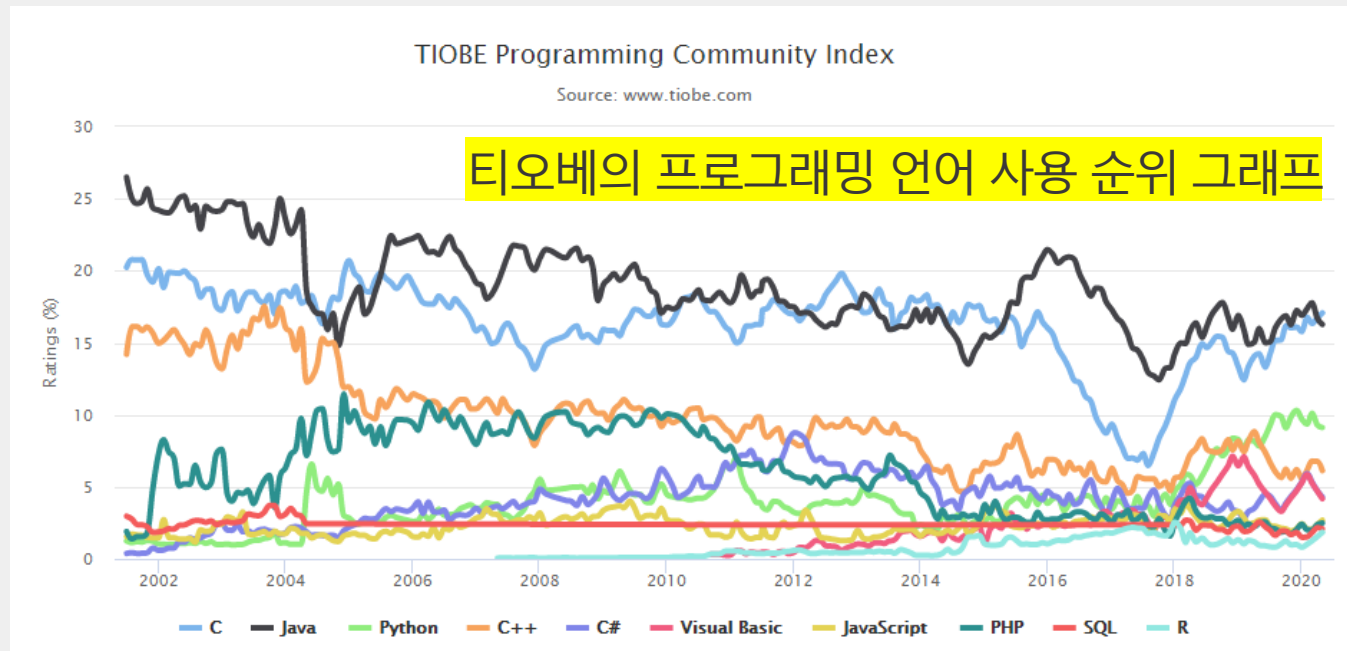
Section 03 제 4차 산업혁명 시대, 모두에게 필요한 파이썬

Chapter 1 도전 프로그래밍

1.1 파이썬 언어란?

파이썬은 배우기 쉽고 누구나 무료로 사용할 수 있는 오픈소스* 프로그래밍 언어이다. 지난 1991년 네덜란드의 귀도 반 로섬이 개발했으며, 현재는 비영리 단체인 파이썬 소프트웨어 재단이 관리하고 있다.

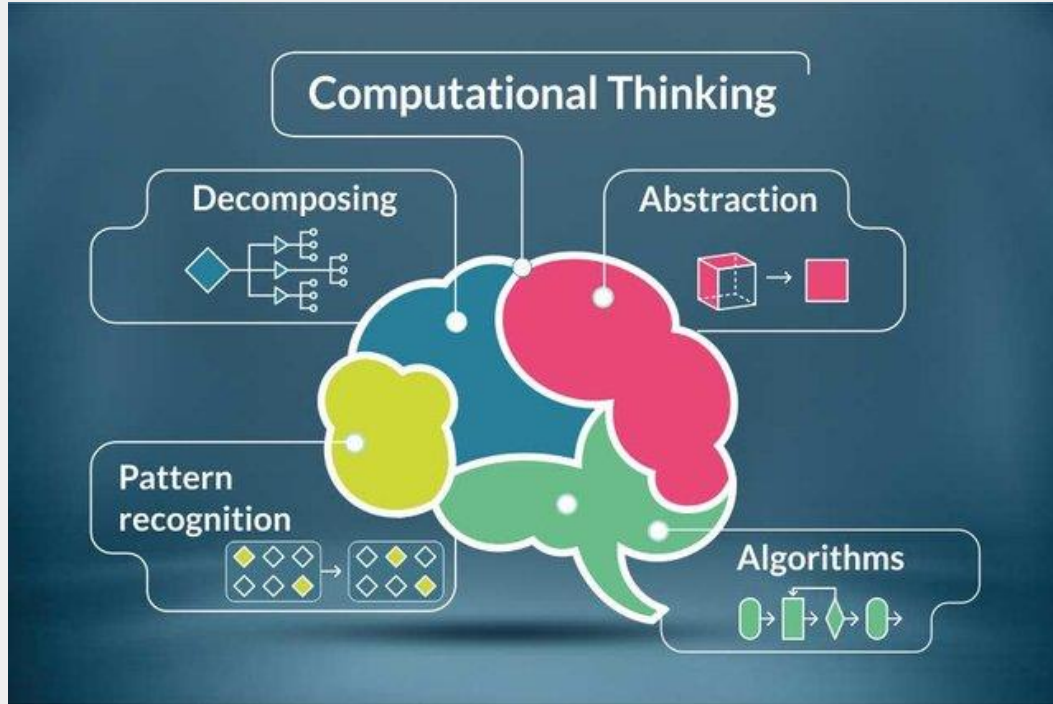
현재 미국과 우리나라의 대학 등 전 세계적으로 가장 많이 가르치는 프로그래밍 언어 중 하나이다. 티오베에서 집계된 프로그래밍 언어 사용 순위에선 C와 자바에 이은 3위를 기록했다.



*오픈 소스 : 소프트웨어를 만든 프로그램 소스를 모두 공개한 것. 파이썬의 소스는 깃허브에서 볼 수 있다.

1.2 컴퓨팅 사고력과 파이썬

컴퓨팅 사고력은 실생활 및 다양한 학문 분야의 문제를 컴퓨터 과학과 컴퓨팅 시스템을 이용해 창의적 해법을 구현해 적용할 수 있는 능력을 말한다. 이 능력에는 추상화 능력, 자동화 능력, 창의.융합 능력이 포함된다.



왼쪽 그림과 같이 컴퓨팅 사고력은 네가지 구성 요소로 이루어진다.

분해 (decomposing)

데이터, 프로세스 또는 문제를 작고 관리가능한 부분으로 나눔

패턴 인식 (Pattern recognition)

데이터의 패턴, 추세 및 정규성을 관찰

추상화 (abstraction)

패턴을 생성하는 일반 원칙을 규정

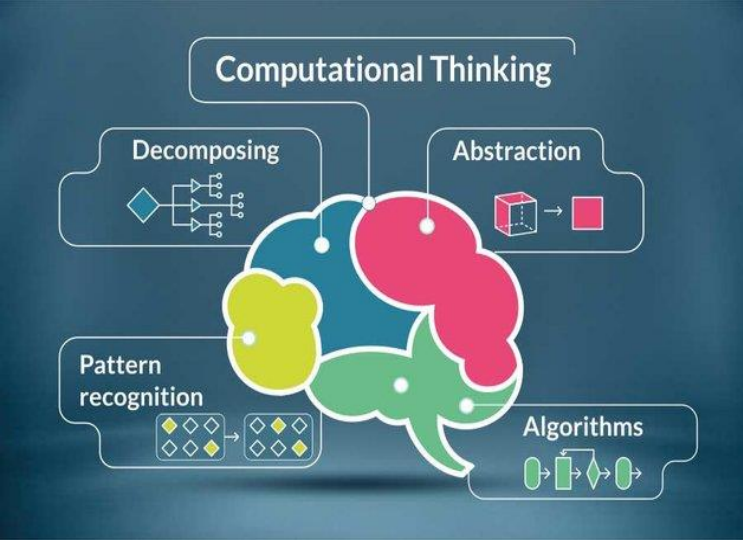
알고리즘 설계 (Algorithms)

이 문제와 유사한 문제 해결을 위한 단계별 지침을 개발

1.2 컴퓨팅 사고력과 파이썬

컴퓨팅 사고력의 향상에 중요한 것은 직접 프로그래밍 해보는 것이다.

책에 정의되어 있는 프로그래밍은 이해 설계 구현 공유의 절차인 UDIS 를 따른다. (코딩 절차와 컴퓨팅 사고력 요소)

코딩 절차	컴퓨팅 사고력 요소 활용	내용
이해U	 <p>The diagram illustrates the components of Computational Thinking. At the top is 'Computational Thinking'. Below it are four interconnected elements: 'Decomposing' (represented by a diamond and branching lines), 'Abstraction' (represented by a 3D cube and a 2D square), 'Pattern recognition' (represented by a sequence of diamonds), and 'Algorithms' (represented by a flowchart with arrows). These elements are arranged around a central brain-like shape.</p>	주어진 문제를 이해하고 파악 분할
설계D		패턴 인식 추상화 알고리즘
구현I		문제 해결을 위해 파이썬으로 코드 개발 실행과 테스트, 디버깅 과정을 거치면서 코 드를 수정하고 필요하면 다시 설계
공유S	협업과 평가 (Collaboration & Evaluation)	자신이 구현한 프로그램을 발표하고 비교 프로그램 개선 연구 피드백 및 평가

2.1 파이썬 개발 도구 설치와 파이썬 쉘의 실행 & 2.2 파이썬 쉘에서 첫 대화형 프로그래밍

Python.org 웹 사이트에 들어가서 설치할 수 있다.

파이썬을 설치했다면 파이썬 쉘 IDLE을 윈도우 실행창에서 찾아 실행할 수 있다.

종료는 File – Exit 또는 단축기 Ctrl + Q로 종료할 수 있다.

파이썬 쉘 IDLE의 상단에는 버전 정보와 help, copyright 등을 이용할 수 있다는 메시지와 프롬프트가 표시되며, 커서가 깜빡인다.

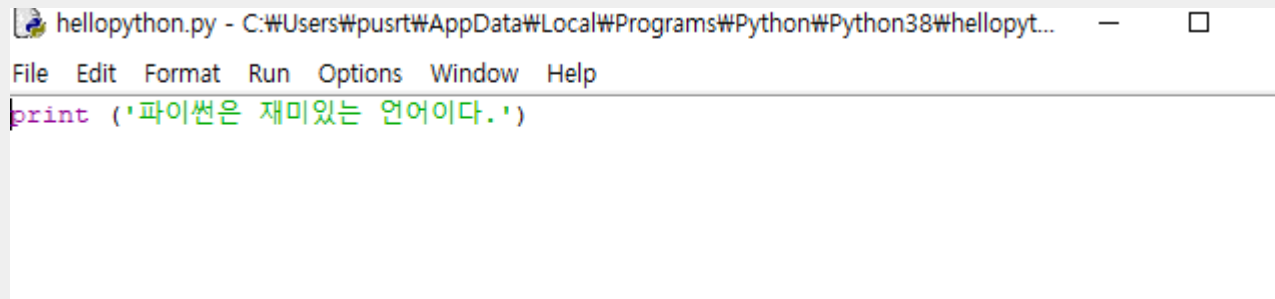
프롬프트는 쉘의 명령이나 파이썬 문장을 기다린다는 의미를 가진 기호이며 커서는 개발자의 명령어 입력을 기다린다.

```
>>> print('Hello, World!')
Hello, World!
>>> |
```

>>>모양의 프롬프트와, | 모양의 커서를 볼 수 있다. 이 창을 "대화형 창" 또는 REPL(read-eval-print-loop)창이라 한다. 작은 따옴표를 붙여 표현하는 문자열(string) Hello, World!를 print('Hello, World!') 로 입력하면 위 결과창과 같이 나온다. 모든 명령어는 첫 칸부터 입력해야 하고 그 이상을 공백으로 두고 입력하면 SyntaxError가 발생한다.

2.3 편집기에서 첫 파일 프로그래밍

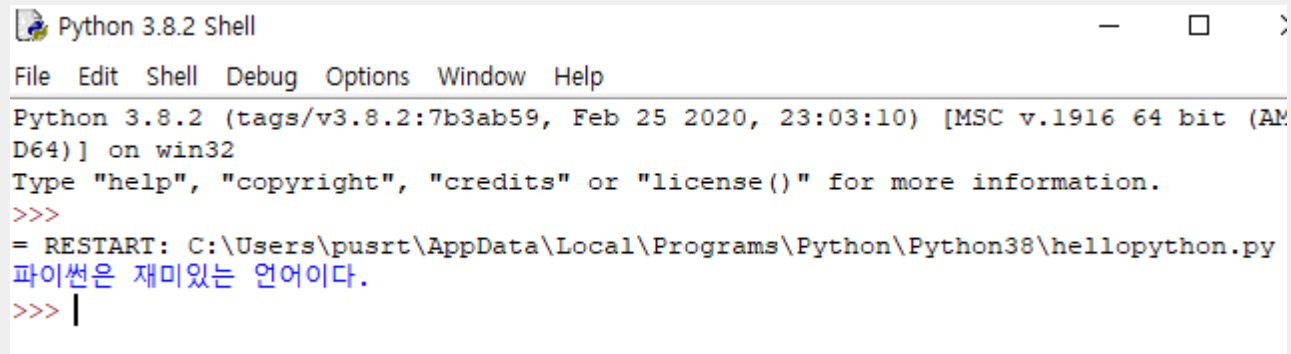
대화형 창을 이용하는 것이 아니라 파이썬 쉘이 제공하는 편집기를 이용해 소스 파일을 저장한 후 실행하는 방식도 있다.



```
hellopython.py - C:\Users\pusrt\AppData\Local\Programs\Python\Python38\hellopyt...
File Edit Format Run Options Window Help
print ('파이썬은 재미있는 언어이다.')
```

파이썬 메뉴의 File – New File 을 선택해 위와 같은 편집기를 나타나게 하고, 소스 파일을 모두 작성하면 저장한다.

저장한 소스 파일을 F5 혹은 Run – Run Module 로 실행하면 아래처럼 소스 파일의 실행 결과가 파이썬 쉘에 표시된다.



```
Python 3.8.2 Shell
File Edit Shell Debug Options Window Help
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 23:03:10) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\pusrt\AppData\Local\Programs\Python\Python38\hellopython.py
파이썬은 재미있는 언어이다.
>>> |
```

파이썬 쉘을 종료하려면 File – Exit 이나 단축키 Ctrl + Q를 사용한다. 쉘에서 명령어 exit() 으로도 종료할 수 있다.

3.1 쉽고 강력한 언어

파이썬의 특징은 다음과 같다.

C나 자바에 비해 간결하고, 인간의 사고 체계와 닮아 사용하기가 쉽다.

무료이며, list, tuple, map 등과 같은 다양한 자료 구조의 제공으로 생산성이 높다.

라이브러리가 독보적이고 강력하며, 데이터과학과 머신 러닝 등과 같은 다양한 분야에 적용할 수 있다.

순수한 자체 언어로 프로그램을 개발하는 것 뿐만 아니라 다른 모듈을 연결해 사용할 수 있는 풀 언어로도 자주 활용된다.

실제 파이썬은 많은 상용 응용 프로그램에서 *스크립트 언어로 채택되고 있다.

*스크립트 언어: 기존에 이미 존재하는 소프트웨어를 제어하기 위한 용도로 쓰이는 언어이다.

3.2 빅데이터 처리와 머신 러닝 등 다양한 분야에 적합한 언어

현재 파이썬은 일반 응용 프로그램의 개발뿐 아니라 인공지능과 빅데이터 처리에 적합한 언어로 인정받아, 여러 실무에 적합한 언어로 엄청난 인기를 끌고 있다. 파이썬의 외부에 풍부한 라이브러리가 있어 다양한 용도로 확장하기 좋기 때문이다.

파이썬은 제 4차 산업혁명 시대의 핵심 기술인 인공지능의 구현과 빅데이터 분석 처리 등의 컴퓨팅 과학 실무 분야에 적합하기 때문에 각광을 받고 있다.

파이썬은 머신러닝과 딥러닝, 빅데이터 처리를 위한 통계 및 분석 방법의 라이브러리도 풍부하게 제공한다.

그러나 대부분의 인터프리터 언어가 그렇듯 실행속도가 느리고, 모바일 앱 개발 환경에는 아직 사용하기 힘들다.

3.3 다양한 종류의 파이썬과 개발 환경

파이썬은 다양한 버전으로 발전했으며, 사실상의 표준은 C언어로 구현된 재단에서 발표하는 C 파이썬이다.

파이썬 개발 환경을 제공하는 개발 도구를 세 가지로 분류하면 다음과 같다:

비주얼 스튜디오나 이클립스 같은 기본 IDE에 파이썬 도구를 설치해 파이썬을 개발

Pycharm, Spyder, Jupyter Notebook 등과 같은 파이썬 전용 IDE를 사용해 개발

Sublime Text, VS Code, Notepad++ 등과 같은 편집기 중심의 개발 환경으로 개발

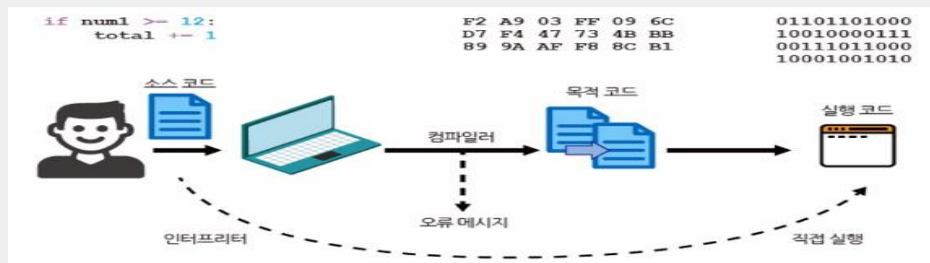
3.4 인터프리트 방식의 언어, 파이썬

파이썬은 인터프리터(해석기) 위에서 실행되는 인터프리트 방식의 언어다.

자바와 C는 컴파일 방식의 언어로, 파이썬과는 프로그램을 기계어로 번역하는 방식이 다르다.

파이썬과 같은 인터프리트 방식의 언어는 동시 통역처럼 한 문장을 한 줄마다 즉시 번역해 실행한다.

자바와 같은 컴파일 방식의 언어는 여러 문장을 소스 단위로 번역해 기계어 파일의 실행 파일을 만든 후 실행한다.



인터프리터와 컴파일러의 차이

Chapter 1 도전 프로그래밍

1. 파이썬 IDLE 에서 다음을 출력하는 코드를 작성하시오.

```
>>> print ('안녕, 파이썬!')
안녕, 파이썬!
>>> |
```

2. 파이썬 셸에서 다음을 출력하는 프로그램을 지정된 파일에 저장해 실행하시오.

helloworld.py - C:\Users\pusrt\AppData\Local\Programs\Python\Python38\helloworld.py

Python 3.8.2 Shell

File Edit Format Run Options Window Help

print ('파이썬은 재미있는 언어이다.')

Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 23:03:10) [MSC v.1916 64 bit (AMD64)] on win32

Type "help", "copyright", "credits" or "license()" for more information.

>>>

= RESTART: C:\Users\pusrt\AppData\Local\Programs\Python\Python38\helloworld.py

파이썬은 재미있는 언어이다.

>>> |

3. 파이썬 셸에서 자신을 소개하는 프로그램을 지정된 파일에 저장해 실행하시오.

introduce.py - C:\Users\pusrt\AppData\Local\Programs\Python\Python38\introduce.py

Python 3.8.2 Shell

File Edit Shell Debug Options Window Help

print('이름 : 홍길동')

print('대학 : 한국대학교')

print('전공 : 컴퓨터공학과')

Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 23:03:10) [MSC v.1916 64 bit (AMD64)] on win32

Type "help", "copyright", "credits" or "license()" for more information.

>>>

== RESTART: C:\Users\pusrt\AppData\Local\Programs\Python\Python38\introduce.py ==

이름 : 홍길동

대학 : 한국대학교

전공 : 컴퓨터공학과

>>> |

Chapter 2

파이썬 프로그래밍을 위한 기초 다지기



Section 01 다양한 자료 : 문자열과 수

Section 02 변수와 키워드 , 대입 연산자

Section 03 자료의 표준 입력과 자료 변환 함수

Chapter 2 도전 프로그래밍

1.1 자료의 종류와 문자열 표현

일상생활에서 흔히 사용하는 글자와 숫자, 날짜 등이 컴퓨터가 처리해야 할 자료이다.

파이썬에선 문자 하나 또는 문자가 모인 단어나 문장, 단락을 문자열(String) 이라 한다. 작은따옴표나 큰따옴표를 써서 표현. 따옴표로 둘러싼 숫자도 문자열로 취급한다.

출력할 때는 문자열이나 숫자 등의 자료를 콘솔에 출력하는 일을 수행하는 *함수 print()를 사용한다.

```
>>> 'P'
'P'
>>> "python"
'python'
>>> '27'
'27'
>>> "3.14"
'3.14'
```

대화형 모드에서 문자열을 표현하면 작은따옴표의 문자열이 표시된다.
파일로 저장하면 출력되지 않는다.

```
>>> print('Hello World!')
Hello World!
>>> print("Hello World!")
Hello World!
>>> print("Hello World!")
SyntaxError: EOL while scanning string literal
>>> |
```

작은따옴표와 큰따옴표 모두 사용할 수 있으나, 앞뒤의 따옴표가 다르면 구문 오류인 SyntaxError가 발생한다. 뒤에 큰따옴표가 빠져 문자열이 끝나지 않고 계속되므로 줄의 끝인 EOL(End of Line)까지 갔다는 구문 오류다.

*함수: 함수는 코딩에서 콘솔에 자료를 출력하는 작업을 도와주는 역할을 한다. 자세한 것은 나중에 배운다.

1.2 문자열 연산자 +, *와 주석

“문자열” ‘연결’ 과 같이 여러 문자열을 계속 쓰는 방법을 사용해 문자열을 쉽게 연결할 수 있다. 공백이 있어도 된다

```
>>> print("python " + 'program')
python program
```

더하기 기호인 +는 문자열을 연결하는 역할을 한다.

```
python program
>>> print("방가 " * 3)
방가 방가 방가
>>> print(4 * '쿨룩 ')
쿨룩 쿨룩 쿨룩 쿨룩
>>> print('파이썬' * '언어')
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    print('파이썬' * '언어')
TypeError: can't multiply sequence by non-int of type 'str'
... |
```

* 는 문자열을 지정된 수만큼 반복한다.

* 를 사용할 때 앞뒤의 순서는 상관없이 정수와 문자열이면 가능하다.

반복 횟수인 정수가 하나 있어야 하므로 문자열로만 반복 연산자를 사용하면 시퀀스를 곱할 수 없다는 오류가 발생한다.

```
>>> print(''''String operator + and * are very easy!
!!!''')
String operator + and * are very easy!
!!
... |
```

작은 따옴표나 큰 따옴표를 연속으로 앞뒤에 3개씩 사용하면 긴 문장이나 여러 줄에 걸쳐 문자열을 처리할 때 편리하다.

```
>>> print('# 이후는 주석') # 한 줄에서 문장 이후에도 주석 사용 가능
# 이후는 주석
... |
```

#는 주석으로, 소스 설명이며 소스 설명이나 파일 이름 등을 담을 때 사용한다.

1.3 정수와 실수의 이해

숫자는 간단히 정수(integer)와 실수(float)로 나뉜다. 소수점이 없는 수는 정수, 있는 수는 실수이다.

이러한 수 역시 `print()`로 출력할 수 있다.

다른 정수 앞에 0이 나오면 구문 오류가 발생한다. 실수는 상관 없다.

프로그래밍 언어에서 실수를 부동소수라 한다. 실수는 소수점의 위치를 이동시킨 실수와 지수승의 곱으로 표현할 수 있기 때문이다.

```
>>> 2.7834e4
27834.0
>>> 2.7834e-4
0.00027834
>>> |
```

실수는 이와 같이 문자 `e`를 사용해 지수승으로 표현할 수 있다. 대문자 `E`도 가능하다. 이때 `e`는 10승을 의미한다.

복소수는 실수보다 상위 개념의 수로, 수학에서 $3+5i$ (허수)와 같은 수다. 다만 파이썬에서는 `i`대신 `j`를 사용한다

```
>>> 3 + 5j + 5 - 2j
(8+3j)
```

복소수의 연산은 이와 같이 이루어진다. `Real`로 실수부, `imag`로 허수부를 참조할 수 있다. 또한 함수 `conjugate()`로 켤레복소수를 반환받을 수 있다.

1.4 정수와 실수의 다양한 연산

수의 연산인 +로 더하기, -로 빼기, *로 곱하기, /로 나누기를 사용할 수 있다. 이러한 연산자를 산술연산자라고 한다. //로 정수 나눗셈을 할 수 있다. 몫과 나머지 연산자는 다양하게 활용할 수 있다.

```
>>> 8 / 5
1.6
>>> 8 // 5
1
```

정수 나눗셈을 하면 몫과 같이 소수 없이 정수만 남는다. 정수와 실수 모두 가능하다.

```
>>> 5 % 3
2
>>> 5 ** 3
125
>>> |
```

나머지 연산자 %는 나눈 나머지가 결과로 나온다.

5 / 3 의 몫이 1이고 나머지가 2이므로, %를 사용하면 나머지인 2가 결과로 나온다.

연속한 별표 2개인 연산자 **은 거듭제곱 연산자이다. $5^{**}3 = 5 * 5 * 5 = 125$

계산의 우선순위는 괄호 계산 > 거듭제곱 > 부호 > *, /, //, % > +, - 순이다.

```
>>> 2 ** 2 ** 3
256
>>> 2 ** ( 2 ** 3)
256
>>> (2 ** 2) ** 3
64
```

다른 계산과 다르게 거듭제곱은 오른쪽에서 왼쪽으로 계산된다.

연산자 ** 는 부호보다 먼저 계산되며, 부호는 ** 를 제외한 연산자보다 먼저 계산된다.

```
>>> 60
60
>>> 3 * _
180
```

대화형 모드에서 마지막에 실행된 결과값은 특별한 저장 공간인 _에 대입된다.

1.4 정수와 실수의 다양한 연산

print()함수는 콤마로 구분해 여러 자료를 출력할 수 있다.

```
>>> print('23000원은', '5000원', 23000//5000, '개')
23000원은 5000원 4 개
>>> |
```

자료 사이에는 빈 공백 문자가 자동으로 삽입된다.

```
>>> eval('3 + 15 / 2')
10.5
```

eval()함수를 사용하면 연산 문자열을 실행한 결과를 반환 받을 수 있다.

2.1 자료형과 type 함수

정수와 실수, 문자열 등을 자료형이라고 한다. 정수는 int, 실수는 float, 문자열은 str로 사용한다. 대화형 모드에서 자료형을 직접 알아보려면 type() 함수를 사용해야 한다.

```
>>> type(3)
<class 'int'>
>>> type(3 + 4j)
<class 'complex'>
```

파이썬은 모든 자료형이 클래스다. 정수의 클래스는 int이다. 복소수는 complex 클래스이다.

2.2 변수와 대입연산자

변수는 자료를 담을 수 있는 그릇이다. 값을 변수에 저장하기 위해선 대입 연산자 = 가 필요하다. =는 오른쪽 값을 왼쪽 변수에 저장하라는 의미이며, 왼쪽에는 반드시 변수가 와야 한다.

```
>>> print(keyword.kwlist)
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'clas
s', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from',
'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass',
'raise', 'return', 'try', 'while', 'with', 'yield']
```

프로그래밍 언어가 이미 정해 놓은 키워드 33개

변수를 구성하는 문자는 영문, 숫자, _를 사용할 수 있으며 숫자는 맨 앞에 올 수 없고 키워드는 사용 불가하다.

2.2 변수와 대입연산자

수학과는 다르게 =가 대입 연산자이므로 `value = value + 1` 같은 연산자를 사용해 `value`의 값을 1 증가시킬 수 있다.

더하기 뿐만 아니라 다른 산술 연산자도 대입 연산자를 사용할 수 있다.

다양한 산술 연산자

연산자	명칭	의미	우선순위	예
+	더하기(add)	두 피연산자(operand)를 더하거나 수의 부호	4, 2	<code>4 + 5</code> , <code>+3</code>
-	빼기(subtract)	두 피연산자를 빼거나 수의 부호	4, 2	<code>9 - 5</code> , <code>-7</code>
*	곱하기(multiply)	두 피연산자 곱하기	3	<code>3 * 4</code>
/	나누기(divide)	왼쪽을 오른쪽 피연산자로 나누기	3	<code>10 / 4</code>
%	나머지(modulus)	왼쪽을 오른쪽 피연산자로 나눈 나머지(remainder)	3	<code>21 % 4</code>
//	몫 나누기(floor division)	왼쪽을 오른쪽 피연산자로 나눈 결과에서 작거나 같은 정수	3	<code>10 // 3</code>
**	거듭제곱, 지수승(exponent)	왼쪽을 오른쪽 피연산자로 거듭제곱	1	<code>2 ** 3</code>

2.2 변수와 대입연산자

```
>>> a, b = 5, 9
>>> print(a,b)
5 9
```

코마로 구분 된 여러 변수에 순서대로 값을 대입할 수 있다.

```
>>> a, b = 5, 9
>>> temp = a
>>> a = b
>>> b = temp
>>> print(a, b)
9 5
>>> |
```

이를 이용하여 간단하게 두 변수에 저장된 값을 교환할 수 있다.

```
>>> distance = 384400
>>> unit = 10000
>>> manUnit, remainder = divmod(distance, unit)
>>> print('지구에서 달까지의 거리: ', manUnit, '만', remainder, '킬로미터')
지구에서 달까지의 거리: 38 만 4400 킬로미터
```

divmod()함수는 나누기 몫 연산과 나머지 연산을 함께 수행해 2개의 결과를 반환한다.

코드를 보면, distance를 unit으로 나눈 몫 연산이 distance에, 나머지 연산 값이 remainder에 저장됐다.

3.2 16진수, 8진수, 2진수 상수 표현

16진법, 8진법, 2진법 등으로 수의 상수를 표현할 수 있다. 16진수는 맨 앞에 0x, 8진수는 0o, 2진수는 0b로 시작한다.

함수 input()은 표준 입력을 문자열로 읽어 반환하는 함수이다.

```
>>> bin(10), oct(10), hex(10)
('0b1010', '0o12', '0xa')
```

10진수를 bin()함수로 2진수, oct()함수로 8진수, hex()함수로 16진수로 변환 가능하다.

```
>>> int('25',10)
25
```

함수 int(문자열, 10)은 문자열을 10진수 정수로 변환한다.

```
>>> int('25',10)
25
>>> int('11', 2)
3
>>> int('10', 8)
8
>>> int('1A', 16)
26
```

여러 진수 형태 문자열을 10진수로 바꾸려면 함수 int(문자열, n)을 사용한다.

여기서 n은 변환하려는 문자열 진수의 숫자이다.

문자열의 맨 앞이 0b, 0o, 0x라면

문자열을 각각 자동으로 2진수, 8진수, 16진수로 변환한다.

Chapter 2 도전 프로그래밍

1. 두 문자열을 표준 입력으로 받아 한 줄에 출력하는 프로그램을 작성하시오.

File Edit Format Run Options Window Help

```
first = input('문자열 1: ')
second = input('문자열 2: ')
print(first, second)
```

```
==== RESTART: C:\Users\pusrt\AppData\Local\Programs\Python\Python38\input.py ===
문자열 1: python
문자열 2: 언어
python 언어
```

3. 아메리카노를 주문받아 총 가격을 출력하는 프로그램을 작성하시오.

americano.py - C:\Users\pusrt\AppData\Local\Programs\Python\Python38\americano.py

File Edit Format Run Options Window Help

```
number = int(input('아메리카노 몇 개 주문하세요?'))
print('총 가격은', 3500 * number, '이다.')
```

```
==== RESTART: C:/Users/pusrt/AppData/Local/Programs/Python/Python38/americano.py
아메리카노 몇 개 주문하세요?7
총 가격은 24500 이다.
```

5. 다음 조건을 참고해 섭씨 온도를 입력 받아 화씨 온도로 변환하는 프로그램을 작성하시오.

cels.py - C:\Users\pusrt\AppData\Local\Programs\Python\Python38\cels.py (3.8.2)

File Edit Format Run Options Window Help

```
celc = int(input('온도 입력 >> '))
fahr = celc * 9 / 5 + 32
fahr2 = celc * 2 + 30
print('정확 계산:', '섭씨:', celc, '화씨:', fahr)
print('약식 계산:', '섭씨:', celc, '화씨:', fahr2)
print('차이:', fahr - fahr2)
```

cels.py - C:\Users\pusrt\AppData\Local\Programs\Python\Python38\cels.py (3.8.2)

File Edit Format Run Options Window Help

```
for i in range(20,42,3):
    a = i * (9 / 5) + 32
    b = i * 2 + 30
    print('섭씨: %d 화씨: %.11f 화씨(약식): %d 차이: %.21f' % (i,a,b,b-a))
```

7. 다음에서 설명하는 함수 float.fromhex(str)을 이해하고 두 16진수 실수를 입력 받아 사칙연산을 수행하는 프로그램을 작성

hex.py - C:\Users\pusrt\AppData\Local\Programs\Python\Python38\hex.py (3.8.2)

File Edit Format Run Options Window Help

```
first = input('첫 번째 16진수 실수 입력>> ')
second = input('두 번째 16진수 실수 입력>> ')
hexf = float.fromhex(first)
hexs = float.fromhex(second)
print('실수1:', hexf, '실수2:', hexs)
print('합:', hexf + hexs)
print('차:', hexf - hexs)
print('곱하기:', hexf * hexs)
print('나누기:', hexf / hexs)
```

```
==== RESTART: C:\Users\pusrt\AppData\Local\Programs\Python\Python38\hex.py ===
```

```
첫 번째 16진수 실수 입력>> f
두 번째 16진수 실수 입력>> e.1
실수1: 15.0 실수2: 14.0625
합: 29.0625
차: 0.9375
곱하기: 210.9375
나누기: 1.0666666666666667
```

Chapter 3

일상에서 활용되는 문자열과 논리 연산



Section 01 문자열 다루기

Section 02 문자열 관련 메소드

Section 03 논리 자료와 다양한 연산

Chapter 3 도전 프로그래밍

1.1 문자열 str 클래스와 부분 문자열 참조 슬라이싱

문자의 나열을 의미하는 문자열은 자료형이 class str이다. 문자열 상수는 str의 객체다.

작은 따옴표 내부에 큰 따옴표를 사용 가능하고, 큰 따옴표 내부에 작은따옴표를 사용 가능하다.

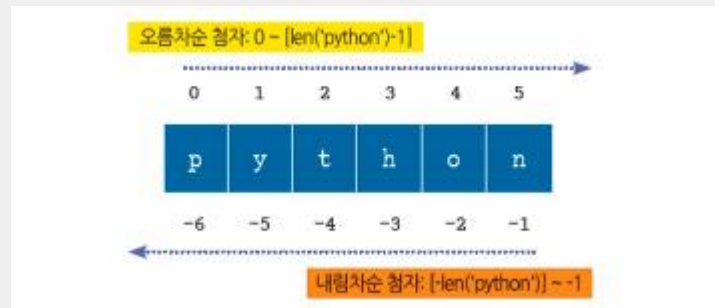
클래스는 객체지향 프로그래밍 언어에서 객체를 정의하기 위한 템플릿이고, 객체는 구체적인 사례인 인스턴스이다.

```
>>> a = 'python'
>>> len(a)
6
```

함수 len()으로 문자열의 길이를 알 수 있다.

문자는 0부터 시작되는 첨자를 기술했다. -1부터 시작돼 -2, -3 으로 작아지는 첨자도 역순으로 참조한다.

문자열의 문자 참조 범위



1.2 문자열의 부분 문자열 참조 방식

문자열의 일부를 참조하는 방법을 슬라이싱 이라고 한다.

[start:end]로 부분 문자열을 start 첨자에서 end-1 첨자까지 반환한다. 음수도 사용할 수 있다.

```
>>> 'python'[1:5]
'ytho'
>>> 'python'[-5:-1]
'ytho'
>>> 'python'[1:-1]
'ytho'
>>> 'python'[5:-1]
''
```

문자열 str에서 end-1 첨자까지 문자열을 반환
음수를 이용한 문자열 슬라이싱
0, 양수, 음수를 혼합한 슬라이싱
start와 end로 구성하는 문자열이 없을 때 반환하는 빈 문자열

```
>>> 'python'[:]
'python'
>>> 'python'[::]
'python'
>>> 'python'[::-1]
'nohtyp'
>>> 'python'[:1]
'p'
```

슬라이싱에서 start와 end를 비우면 문자열 전체를 반환
[start:end:step]으로 문자 간격을 조정할 때 역시 str[:]이나 str[:1]을 이용해 문자열 전체를 반환

```
>>> 'python'[1:5:2]
'yh'
>>> 'python'[::-1]
'nohtyp'
```

문자 사이의 간격을 step으로 조정할 수 있다. 음수도 가능하다.

1.3 문자 함수와 이스케이프 시퀀스

함수 `ord()`로 문자의 유니코드 번호를, 함수 `chr()`로 유니코드 번호에 배정된 유니코드 번호를 알 수 있다.
하나의 문자를 역슬래시(`\`)로 시작하는 조합으로 표현하는 문자를 이스케이프 시퀀스 문자라고 한다.

```
>>> min('ipython')
'h'
>>> max('ipython')
'y'
>>> min('3259')
'2'
>>> max('3259')
'9'
>>> max(3, 96.4, 13)
96.4
>>> min('ipython', 'java')
'ipython'
```

*내장 함수 `min()`과 `max()`는 각각 최댓값과 최솟값을 반환한다.
인자가 문자열 1개이면 문자열을 구성하는 문자에서
코드 값으로 최대와 최소인 문자를 반환한다.

이스케이프 시퀀스 문자	설명
<code>\</code>	역슬래시
<code>\'</code>	작은따옴표
<code>\"</code>	큰따옴표
<code>\a</code>	벨소리(알람)
<code>\b</code>	백스페이스(이전 문자 지우기)
<code>\n</code>	새 줄
<code>\N{name}</code>	유니코드의 이름
<code>\r</code>	동일한 줄의 맨 앞으로 이동 (파이썬 셸에서는 다음 줄로)
<code>\f</code>	폼피드(form feed) (예전 프린터에서 다음 페이지의 첫 줄로 이동)
<code>\t</code>	수평 탭
<code>\w</code>	수직 탭
<code>\uXXXX</code>	16비트 16진수 코드
<code>\UXXXXXXXX</code>	32비트 16진수 코드
<code>\ooo</code>	8진수의 코드 문자
<code>\xhh</code>	16진수의 코드 문자

*내장 함수 : 내장 함수는 파이썬 라이브러리로 인터프리터에서 아무런 설정 없이 바로 사용할 수 있는 함수를 말한다.

2.1 문자열 대체와 부분 문자열 출현 횟수, 문자열 삽입

클래스에 소속된 함수를 메소드라 한다. 메소드의 호출은 '문자열 객체명.함수 이름(인자)'와 같이 사용한다.

```
>>> str = '자바는 인기 있는 언어 중 하나다.'
>>> str.replace('자바는', '파이썬은')
'파이썬은 인기 있는 언어 중 하나다.'
```

함수 `replace()`를 사용하면 왼쪽의 문자를 오른쪽의 문자로 대체해서 반환한다.

```
>>> str = '파이썬 파이썬 파이썬'
>>> str.replace('파이썬', 'python', 3)
'python python python'
```

함수 `replace()`를 (old, new, count)순으로 사용하면 count 횟수 만큼 바꾼다.

```
>>> str = '단순한 것이 복잡한 것보다 낫다.'
>>> str.count('복잡')
1
```

함수 `count()`를 사용하면 문자열에서 문자나 문자열의 출현 횟수를 알 수 있다.

```
>>> num = '12345'
>>> '->'.join(num)
'1->2->3->4->5'
```

함수 `join()`을 사용하면 문자열의 문자와 문자 사이에 원하는 문자열을 삽입할 수 있다.

함수는 특정 작업을 수행하는 독립된 코드 모임이다. 반면, 메소드는 클래스에 포함되어 있는 함수를 말한다.

2.2 문자열 찾기

```
>>> str = '자바 c 파이썬 코틀린'
>>> str.find('자바')
0
>>> str.index('파이썬')
5
```

클래스 str에서 부분문자열 sub가 맨 처음에 위치한 첨자를 반환 받으려면 str.find 또는 str.index를 사용한다.
함수 index()는 찾는 문자열이 없으면 에러를 발생시키지만 find()는 -1을 반환한다.

```
>>> 'python ipython'.rfind('on')
12
>>> 'python ipyhon'.rindex('py')
8
```

메소드 rfind()와 rindex()는 역순으로 문자열을 찾아 반환한다.

2.3 문자열 나누기

```
>>> '사과 배 복숭아 딸기 포도'.split()
['사과', '배', '복숭아', '딸기', '포도']
>>> '데스크톱 100000, 노트북 1800000, 스마트폰 1200000'.split(',')
['데스크톱 100000', ' 노트북 1800000', ' 스마트폰 1200000']
```

메소드 split()은 문자열에서 공백을 기준으로 문자열을 나눠 준다.
괄호 안에 문자열 값이 있으면 이 문자열을 이용해 문자열을 나눈다.

2.4 다양한 문자열 변환 메소드

모두 대문자로 변환해 반환하는 upper(), 모두 소문자로 변환하는 lower(), 폭을 지정하고 중앙에 문자열을 배치하는 center(), 폭을 지정하고 정렬하는 메소드 ljust() rjust(), 특정 문자를 제거하는 strip(), 지정한 폭 앞에 0을 채워 넣는 zfill()등 다양한 메소드가 있다.

2.5 출력을 정형화 하는 함수 format()

메소드 strformat을 사용해 문자열 중간중간에 변수나 상수를 출력할 수 있다.

메소드 format()을 호출한 문자열을 print()함수의 인자로 사용하면 원하는 결과가 표시된다.

인자의 순서를 나타내는 정수를 0부터 삽입하면 출력 순서를 조정할 수 있다.

```
>>> print ('{0} / {1} = {2}'.format(a, b, a/b))  
10 / 4 = 2.5
```

```
>>> print('{0:d} / {1:d} = {2:f}'.format(a, b, a/b))  
10 / 3 = 3.333333
```

출력 폭(5, 10 등) 과 출력 형식(d, f)등을 지정할 수 있다.

2진수와 8진수, 16진수로 출력하려면 각각 b, o, x 로 유형을 지정해 출력할 수 있다.

2.6 C언어의 포매팅 스타일인 %d와 %f등으로 출력

C언어에서 사용하는 형식 지정사 %d, %x, %o, %f, %c, %s등을 사용해 출력할 수 있다. %%는 %를 출력한다.

```
>>> '%d - %x = %o' % (30, 20, 30-20)  
'30 - 14 = 12'  
>>> print('%10.2f' % 2.718281)  
2.72
```

출력 인자가 1개이면 괄호는 생략할 수 있다.

3.1 논리 값과 논리 연산

논리 값으로 참을 뜻하는 True와 거짓을 뜻하는 False를 키워드로 제공한다. 이 값의 자료형은 bool이다.

정수의 0, 실수의 0.0, 빈 문자열 등은 false이며, 음수와 양수, java등의 문자열은 True이다.

내장함수 bool로 인자의 논리값을 알 수 있다.

논리곱인 and과 &는 두 항이 모두 참이어야 True이고, 논리합인 or과 |는 두 항이 모두 거짓이어야 False 이다.

배타적 논리합인 ^은 두 항이 다르면 True, 같으면 False이다. Not은 뒤에 위치한 논리값을 바꾼다.

논리 연산은 not연산, 논리곱, 논리합 순이다.

대소 비교 관계 연산자

3.2 관계 연산

관계 연산자는 수나 문자열 등의 크기 비교에 사용된다.

결과는 논리값이며,

문자열은 문자를 왼쪽부터 유니코드 값으로 비교한다.

True와 False를 1과 0으로 산술 연산에 활용할 수 있다.

연산자	연산 사용	의미	문자열 관계 연산
>	a > b	크다(greater than).	사전 순서(lexicographically)에서 뒤에(코드 값이 크다)
>=	a >= b	크거나 같다 (greater than or equal to).	사전 순서에서 뒤에(코드 값이 크다) 있거나 동일
<	a < b	작다(less than).	사전 순서에서 앞에(코드 값이 작다)
<=	a <= b	작거나 같다 (less than or equal to).	사전 순서에서 앞에(코드 값이 작다) 있거나 동일
==	a == b	같다(equal to).	사전 순서에서 동일
!=	a != b	다르다(not equal to).	사전 순서와 다름

3.3 멤버십 연산 in

멤버십 연산 in은 멤버의 소속을 알 수 있는 연산으로, 결과는 True, False의 논리 값이다.

3.4 비트 논리 연산

비트 연산은 비트와 비트의 연산을 말한다.

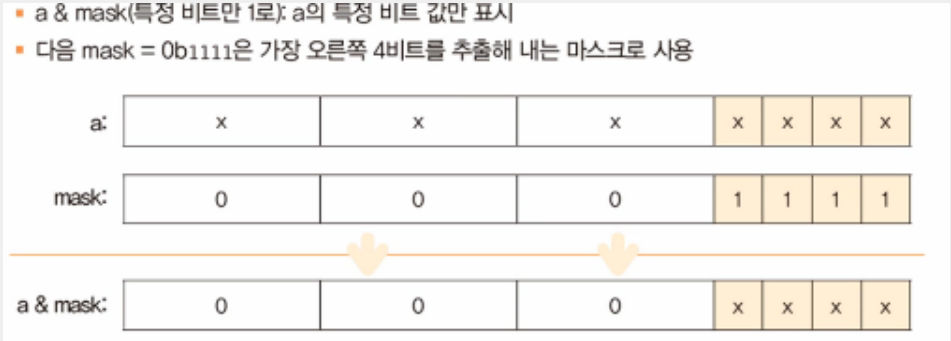
이를 이해하려면 정수를 2진수 비트로 변환해 비트 연산을 수행한 후 그 결과를 10진수로 변환해야 한다.

연산	결과
x in s	문자열s에 부분문자열 x가 있으면 True, 없으면 False
x not in s	문자열s에 부분문자열 x가 있으면 True, 없으면 False

연산식	10진수	2진수 표현								설명
		128	64	32	16	8	4	2	1	
a	23	0	0	0	1	0	1	1	1	23의 2진수 00010111
b	57	0	0	1	1	1	0	0	1	57의 2진수 00111001
a & b	17	0	0	0	1	0	0	0	1	비트가 모두 1이면 1
a b	63	0	0	1	1	1	1	1	1	비트가 하나라도 1이면 1
a ^ b	46	0	0	1	0	1	1	1	0	두 비트가 다르면 1, 같으면 0

논리합, 논리곱, 배타적 논리합의 결과는 논리 연산에서 True를 1, False를 0으로 생각하면 동일하다.

원하는 특정 비트를 모두 1로 지정한 mask값을 정수a와 논리곱으로 연산한 a & mask의 결과는 a의 특정 비트 값만을 뽑아 낸다.



3.4 비트 논리 연산

비트 배타적 논리합의 $(a \wedge b) \wedge b == a \wedge (b \wedge b) == a \wedge 0 == a$ 특성을 활용해 \wedge 연산으로 암호화에 활용할 수 있다. 정수 a 를 key 와 연산식 $a \wedge key$ 결과를 저장한 후, 다시 이 저장된 암호화 결과인 $a \wedge key$ 를 사용해 $a \wedge key \wedge key$ 하면 원래 a 로 복호화 할 수 있다.

3.5 비트 이동 연산

비트 이동 연산자 \gg 와 \ll 는 연산자의 방향으로 지정된 횟수만큼 이동시키는 연산자이다.

\ll 에 의해 생기는 오른쪽 빈 자리는 모두 0으로 채워지며, \gg 에 의한 빈 자리는 0이나 양수이면 0, 음수이면 1이 채워진다.

오른쪽 이동 연산 수행 1회마다 2로 나눈 효과가 있고, 왼쪽 이동 연산 1회마다 2를 곱한 효과가 있다.

```
>>> print( 20 + 30 // 5 >> 2 | 1)
7
>>> print ( ((20 + ( 30 // 5 )) >> 2 ) | 1)
7
>>> print(3 << 2 < 100 and 3 ** 3 >> 2 >=5)
True
>>> print((3 << 2 < 100) and (((3 ** 3 ) >> 2) >=5))
True
```

파이썬은 지수, 단항, 산술, 비트, 관계, 비교, 논리 연산 순으로 연산된다.

Chapter 3 도전 프로그래밍

1. 한 줄의 문자열을 표준 입력으로 받아 문자열의 소속 문자를 참조할 범위를 출력하고 다시 첨자 하나를 입력받아 문자열의 전체와 길이 그리고 참조 문자를 출력하는 프로그램을 작성하시오.

string.py - C:\Users\pusrt\AppData\Local\Programs\Python\Python38\string.py (3.8.2)

File Edit Format Run Options Window Help

```
string = input('문자열 입력>>')
print('참조할 첨자: 0~', len(string)-1)
number = int(input('참조할 첨자 입력>>'))
print('문자열 :', string, '길이 :', len(string))
print('참조 문자 :', string[number])
```

```
>>>
문자열 입력>>Python is a good language!
참조할 첨자: 0~ 25
참조할 첨자 입력>>12
문자열 : Python is a good language! 길이 : 26
참조 문자 : g
>>> |
```

3. 파이썬의 첫 번째 철학을 저장한 후, `replace()`를 사용해 두 번째 철학으로 다시 저장해 출력하는 프로그램을 작성하시오.

replace.py - C:/Users/pusrt/AppData/Local/Programs/Python/Python38/replace.py (3.8.2)

File Edit Format Run Options Window Help

```
str = 'Beautiful is better than ugly.'
print(str)
print('위 철학을 메소드 replace()를 사용해 다음 철학으로 다시 저장')
print(str.replace('Beautiful', 'Explicit').replace('ugly', 'implicit'))
```

```
Beautiful is better than ugly.
위 철학을 메소드 replace()를 사용해 다음 철학으로 다시 저장
Explicit is better than implicit.
^^^
```

5. 문자열의 `split()`메소드를 사용해 시각 정보를 표준입력으로 받아 입력된 문자열을 출력하고 다시 시, 분, 초로 출력하는 프로그램을 작성하시오.

time.py - C:\Users\pusrt\AppData\Local\Programs\Python\Python38\time.py (3.8.2)

File Edit Format Run Options Window Help

```
time = input('시각 정보(16:30:15) 입력>>')
print('입력 시각 정보:', time)
hours, mins, secs = time.split(':')
print(hours, '시', mins, '분', secs, '초')
```

```
>>>
시각 정보(16:30:15) 입력>>17:29:52
입력 시각 정보: 17:29:52
17 시 29 분 52 초
>>>
```

Chapter 3 도전 프로그래밍

7. -7에서 0까지 10진수와 2진수, 8진수, 16진수를 다음과 같이 정형화해 출력하는 프로그램을 작성하시오.

```
octhex.py - C:/Users/pusrt/AppData/Local/Programs/Python/Python38/octhex.py (3.8.2)
File Edit Format Run Options Window Help
mask = 0xff
print('10진수: {}, 2진수: {:b}, 8진수: {:o}, 16진수: {:x}'.format(-7, -7&mask, -7&mask, -7&mask))
print('10진수: {}, 2진수: {:b}, 8진수: {:o}, 16진수: {:x}'.format(-6, -6&mask, -6&mask, -6&mask))
print('10진수: {}, 2진수: {:b}, 8진수: {:o}, 16진수: {:x}'.format(-5, -5&mask, -5&mask, -5&mask))
print('10진수: {}, 2진수: {:b}, 8진수: {:o}, 16진수: {:x}'.format(-4, -4&mask, -4&mask, -4&mask))
print('10진수: {}, 2진수: {:b}, 8진수: {:o}, 16진수: {:x}'.format(-3, -3&mask, -3&mask, -3&mask))
print('10진수: {}, 2진수: {:b}, 8진수: {:o}, 16진수: {:x}'.format(-2, -2&mask, -2&mask, -2&mask))
print('10진수: {}, 2진수: {:b}, 8진수: {:o}, 16진수: {:x}'.format(-1, -1&mask, -1&mask, -1&mask))
print('10진수: {}, 2진수: {:b}, 8진수: {:o}, 16진수: {:x}'.format(0, 0, 0, 0))
|
```

```
10진수: -7, 2진수: 11111001, 8진수: 371, 16진수: f9
10진수: -6, 2진수: 11111010, 8진수: 372, 16진수: fa
10진수: -5, 2진수: 11111011, 8진수: 373, 16진수: fb
10진수: -4, 2진수: 11111100, 8진수: 374, 16진수: fc
10진수: -3, 2진수: 11111101, 8진수: 375, 16진수: fd
10진수: -2, 2진수: 11111110, 8진수: 376, 16진수: fe
10진수: -1, 2진수: 11111111, 8진수: 377, 16진수: ff
10진수: 0, 2진수: 0, 8진수: 0, 16진수: 0
>>> |
```

Chapter 4

일상생활과 반복되는 조건과 반복



Section 01 조건에 따른 선택 if else

Section 02 반복을 제어하는 for문과 while문

Section 03 임의의 수인 난수와 반복을 제어하는 break문, continue문

Chapter 4 도전 프로그래밍

Section 01 조건에 따른 if else

1.1 조건의 논리 값에 따른 선택 if

조건에 따라 해야 할 일을 처리해야 하는 경우 if문을 사용한다.

논리 표현식의 결과가 True 이면 if문 이하의 문장들을 실행한다. False이면 실행하지 않는다.

```
>>> height = 152
>>> if 140 <= height:
    print('롤러코스터 T-Express, 즐기세요!')
```

```
롤러코스터 T-Express, 즐기세요!
```

if문에서 논리 표현식 이후에는 반드시 콜론이 있어야 한다.

콜론 이후 다음 줄부터 시작되는 블록은 반드시 들여쓰기를 해야 한다.

들여쓰기는 보통 4개의 빈 공백으로 Tab을 누르면 빈 공백이 삽입된다.

1.2 조건에 따라 하나를 선택하는 if else

if문에서 논리 표현식의 결과가 false이면 else: 이후의 블록을 실행한다.

```
grade = float(input('1학기 평균 평점은? '))
if 3.8 <= grade:
    print('축하합니다! 장학금 지급 대상자이다.')
print('당신의 1학기 평균 평점 %.2f이다.' % (grade))
```

else이후의 콜론도 반드시 필요하다. 이후 들여쓰기와 블록이 필요하다.

문장2 이후에 엔터 키를 누르면 자동으로 열을 맞추지만,

if와 열을 반드시 맞춰 else를 입력해야 한다.

1.3 여러 조건 중에서 하나를 선택하는 if elif

논리 표현식1이 False이면 elif논리 표현식 2로 진행되며, 이후도 마찬가지다. elif는 필요한 만큼 늘릴 수 있다.

마지막 else는 선택적으로 생략할 수 있다.

```
PM = float(input('미세먼지(10마이크로그램)의 농도는 ? '))
if 151 <= PM:
    print('미세먼지 농도: {:.2f}, 등급: {}'.format(PM, '매우 나쁨'))
elif 81 <= PM:
    print('미세먼지 농도: {:.2f}, 등급: {}'.format(PM, '나쁨'))
elif 31 <= PM:
    print('미세먼지 농도: {:.2f}, 등급: {}'.format(PM, '보통'))
else:
    print('미세먼지 농도: {:.2f}, 등급: {}'.format(PM, ' 좋음'))
```

논리 표현식 1

논리 표현식 2

1.4 중첩된 조건

부류를 선택한 후에 구체적인 내용을 선택하는 것을 구현하기 위해선 if문의 내부에 다시 if문을 사용한다.

```
category = int(input('원하는 음료는? 1. 커피 2. 주스 '))
if category == 1:
    menu = int(input('번호 선택? 1. 아메리카노 2. 카페라테 3. 카푸치노 '))
    if menu == 1:
        print('1. 아메리카노 선택')
    elif menu == 2:
        print('2. 카페라테 선택')
    elif menu == 3:
        print('3. 카푸치노 선택')
else:
    menu = int(input('번호 선택? 1. 키위주스 2. 토마토주스 3. 오렌지주스 '))
    if menu == 1:
        print('1. 키위주스 선택')
    elif menu == 2:
        print('2. 토마토주스 선택')
    elif menu == 3:
        print('3. 오렌지주스 선택')
```

2.1 시퀀스의 내부 값으로 반복을 실행하는 for문

반복문을 사용하면 중복을 줄여 프로그램이 간결해지고 효과적이다. 파이썬 에서 제공하는 반복 방법은 두 가지이다.

while문은 <반복조건>에 따른 반복이며, for문은 항목의 나열인 <시퀀스>의 구성 요소가 변수에 저장돼 반복을 수행한다.

```
sum = 0
for i in 1.1, 2.5, 3.6, 4.2, 5.4:
    sum += i
    print(i, sum)
else:
    print('합 : %.2f, 평균: %.2f' % (sum, sum / 5))
```

- 1.여러 개의 값을 갖는 시퀀스 (in의 뒤)에서 변수(for의 뒤)에 하나의 값을 순서대로 할당한다.
- 2.할당된 변수 값을 가지고 문장을 순차적으로 실행한다.
- 3.시퀀스의 마지막 항목까지 값을 순차적으로 할당해 반복 블록을 실행한다.
- 4.마지막까지 실행한 후 선택사항인 else: 블록을 실행하고 반복을 종료한다.

```
>>> for i in range(1, 10, 2):
    print(i, end = ' ')

1 3 5 7 9
```

반복 for)문의 시퀀스에 내장함수 range()를 사용하면 매우 간결하다.

함수 range(start, stop, step)으로 시퀀스의 시작 지점, 시퀀스의 끝 지점, 증가 값을 모두 설정할 수 있다.

내장함수 range()는 일정 간격의 정수 값을 가진 시퀀스를 생성해 주는 함수로, 적어도 1개의 인자를 지정해 줘야 한다. 하나만 지정하면 자동으로 stop값이 지정된다.

2.2 횟수를 정하지 않은 반복에 적합한 while 반복

while문은 for문에 비해 간결하며 논리 표현식의 값에 따라 반복을 수행한다.

횟수를 정해 놓지 않고 조건이 false가 될 때까지 반복을 수행하는 데 적합하다.

```
MAXNUM = 4
MAXHEIGHT = 130

more = True
cnt = 0
while more: 논리 표현식
    height = float(input("키는 ? "))
    if height < MAXHEIGHT:
        cnt += 1
        print('들어가요.', '%d명' % cnt)
    else:
        print('커서 못 들어갑니다.')
    if cnt == MAXNUM:
        more = False
else:
    print('%d명 모두 찾습니다. 다음 번에 이용하세요.' % cnt)
```

반복 몸체

논리 표현식이 True이면 반복 몸체를 실행한 후 다시 논리 표현식을 검사해 실행한다. 논리 실행식이 False이면 반복 몸체를 실행하지 않고 선택 사항인 else: 이후의 블록을 실행한 후 반복을 종료한다.

2.3 중첩된 반복

for문을 중첩으로 사용하여 외부의 반복과 내부의 반복으로 나눠 중첩된 반복을 사용할 수 있다.

3.1 임의인 수인 난수 발생과 반복에 활용

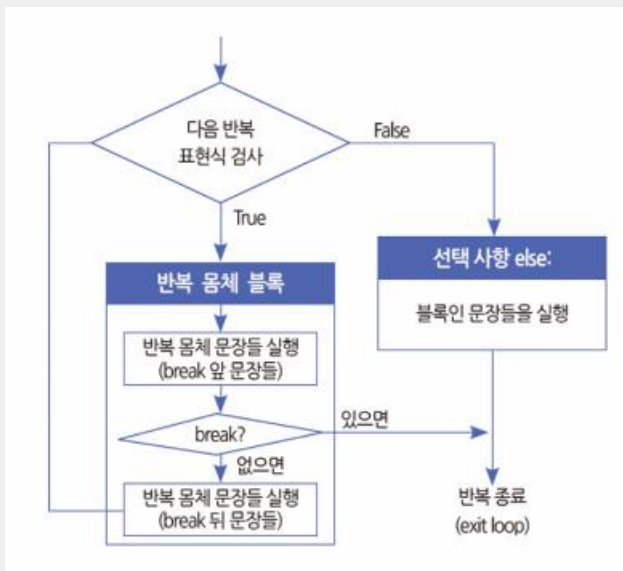
파이썬에서는 모듈 random과 함수 randint(시작, 끝)를 사용해 시작과 끝을 포함한 수 사이에서 임의의 정수를 얻을 수 있다.

3.2 반복을 제어하는 break문과 continue문

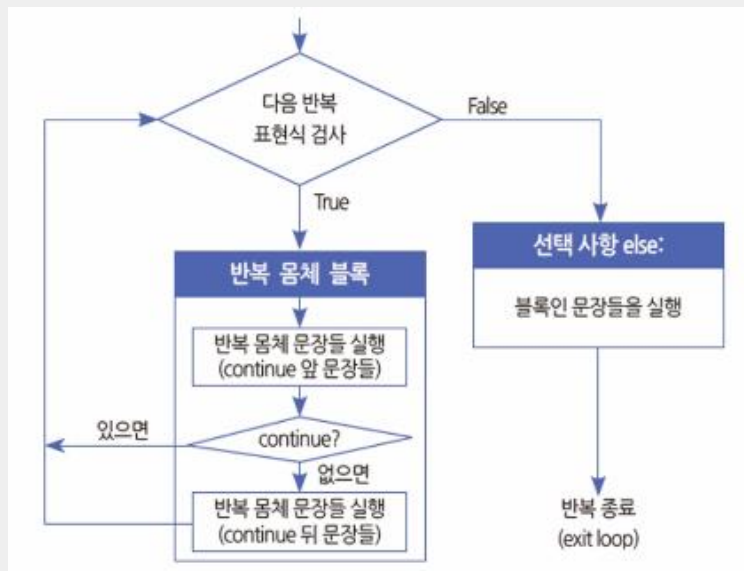
반복 while의 논리 표현식이 True라면 무한히 반복된다.

문장 break는 for이나 while 내부에서 else: 블록을 실행시키지 않고 반복을 무조건 종료한 후 반복 문장 이후를 실행한다.

문장 continue는 이후의 반복 몸체를 실행하지 않고 다음 반복을 위한 논리 조건을 실행한다.



break문의 흐름도



continue문의 흐름도

Chapter 4 도전 프로그래밍

1. 다음을 참고해 월을 표준 입력으로 입력 받아 계절을 출력하는 프로그램을 작성 하시오.

```
month = int(input("월 입력 ? "))

if month <= 3:
    print(month, '월 겨울')
elif month <= 5:
    print(month, '월 봄')
elif month <= 8:
    print(month, '월 여름')
elif month <= 10:
    print(month, '월 가을')
elif month <= 12:
    print(month, '월 겨울')
```

```
월 입력 ? 3
3 월 겨울
>>> |
```

3. 다음을 참고해 1에서 99까지의 난수인 임의의 정수 3개를 대상으로 가장 큰 정수를 출력하는 프로그램을 작성 하시오.

```
randint.py - C:\Users\pusrt\AppData\Local\Programs\Python\Python38\randint.py (3...
File Edit Format Run Options Window Help
from random import randint

a = randint(1,99)
b = randint(1,99)
c = randint(1,99)

if a>=b and a>=c:
    print(a, b, c, '중에서 최대:', a)
elif b>=a and b>=c:
    print(a, b, c, '중에서 최대:', b)
else :
    print(a, b, c, '중에서 최대:', c)
```

```
11 61 82 중에서 최대: 82
>>> |
```

Chapter 4 도전 프로그래밍

5. 다음을 참고해 섭씨 온도 20도에서 41도까지 3도씩 증가하면서 화씨로 변환해 출력하는 프로그램을 작성 하시오.

```
cels.py - C:\Users\pusrt\AppData\Local\Programs\Python\Python38\cels.py (3.8.2)
File Edit Format Run Options Window Help
for i in range(20,42,3):
    a = i * (9 / 5) + 32
    b = i * 2 + 30
    print('섭씨: %d 화씨: %.11f 화씨 (약식): %d 차이: %.21f' % (i,a,b,b-a))
```

```
섭씨: 20 화씨: 68.0 화씨 (약식): 70 차이:2.00
섭씨: 23 화씨: 73.4 화씨 (약식): 76 차이:2.60
섭씨: 26 화씨: 78.8 화씨 (약식): 82 차이:3.20
섭씨: 29 화씨: 84.2 화씨 (약식): 88 차이:3.80
섭씨: 32 화씨: 89.6 화씨 (약식): 94 차이:4.40
섭씨: 35 화씨: 95.0 화씨 (약식): 100 차이:5.00
섭씨: 38 화씨: 100.4 화씨 (약식): 106 차이:5.60
섭씨: 41 화씨: 105.8 화씨 (약식): 112 차이:6.20
>>> |
```

7. 정수 1개를 표준 입력으로 받아 소수인지를 판별하는 프로그램을 작성 하시오.

```
prime.py - C:\Users\pusrt\AppData\Local\Programs\Python\Python38\prime.py (3.8.2)
File Edit Format Run Options Window Help
prime = int(input('소수 (prime number) 인지를 판별한 2 이상의 정수 입력>>'))
is_Prime = '소수이다.'

for i in range(2, prime):
    if prime % i == 0:
        is_Prime = '소수가 아니다.'
        break

print('정수 {}는 {}'.format(prime, is_Prime))
```

```
소수 (prime number) 인지를 판별한 2 이상의 정수 입력>>11
정수 11는 소수이다.
```

Chapter 5

항목의 나열인 리스트와 튜플



Section 01 여러 자료 값을 편리하게 처리하는 리스트

Section 02 리스트의 부분 참조와 항목의 삽입과 삭제

Section 03 항목의 순서나 내용을 수정할 수 없는 튜플

1.1 리스트의 개념과 생성

파이썬 에서 제공하는 여러 항목(item)을 하나의 단위로 묶어 손쉽게 관리하는 복합 자료형 중 대표적인 것이 리스트이다. 리스트는 항목의 나열인 시퀀스이다. 리스트는 콤마로 구분된 항목들의 리스트로 표현되며, 같은 자료형 일 필요는 없다. 항목 순서는 의미가 있으며, 항목 자료 값은 중복해도 상관 없다.

```
coffee = ['에스프레소', '아메리카노', '카페라테', '카페모카']
print(coffee)
print(type(coffee))

num = 0
for s in coffee:
    num += 1
    print('%d. %s' % (num, s))
```

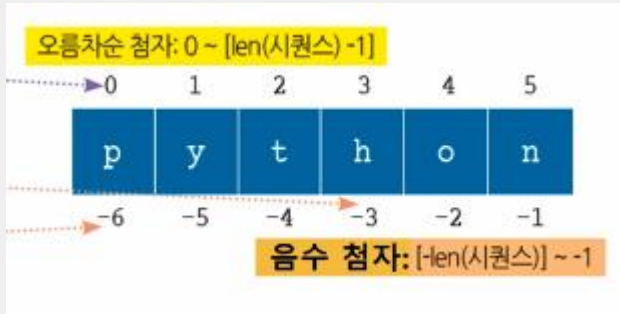
대괄호로 생성하며 변수에 대입할 수 있다. 함수 print(변수이름)으로 손쉽게 리스트를 출력할 수 있다. 자료형은 list이다. 리스트도 시퀀스이므로, 반복for문의 시퀀스 위치에 리스트를 배치해 각 항목을 변수에 저장함으로써 반복을 구현할 수 있다.

```
goods = []
for i in range(3):
    item = input('구입할 품목은 ? ')
    goods.append(item)
    print(goods)
print('길이 : %d' % len(goods))
```

빈 대괄호나 인자가 없는 내장함수 list()를 이용해 빈 리스트를 생성할 수 있다. 리스트의 가장 뒤에 항목을 추가하려면 메소드 append()를 사용하면 된다. 리스트의 항목 수를 알기 위해선 함수 len(리스트)로 항목 수를 알 수 있다.

1.2 리스트의 항목 참조

리스트에서 첨자를 사용해 항목 하나하나를 참조할 수 있다. 이는 문자열에서 배운 첨자 형식과 동일하다.



문자열 python으로 만들어진 리스트는 길이가 6이므로 0~5 그리고 -6~-1로 첨자를 사용할 수 있다.

1.3 리스트의 항목 수정

```
food = ['짜장면', '짬뽕', '우동', '울면']
print(food)
# 탕수육 주문 추가
food.append('탕수육')
print(food)
# 짬뽕을 굴짬뽕을 주문 변경
food[1] = '굴짬뽕'
print(food)

# 우동을 물만두로 주문 변경
food[food.index('우동')] = '물만두'
print(food)
```

리스트의 메소드 count(값)은 값을 갖는 항목의 수를 반환한다.

메소드 index(값)은 인자인 값의 항목이 위치한 첨자를 반환한다.

해당 코드는 리스트의 첨자를 이용한 항목을 대입 연산자의 오른쪽에 위치시켜 리스트의 항목을 수정한 것이다. 항목을 수정할 때는 첨자 뿐만 아니라 메소드 index()도 사용할 수 있다.

Section 01 여러 자료 값을 편리하게 처리하는 리스트

1.4 리스트 내부에 다시 리스트를 포함하는 중첩 리스트

리스트 내부에 다시 리스트가 항목으로 올 수 있다. 해당 항목을 참조하려면 첨자를 두 번 사용해 표시해야 한다.

```
animal = [['사자', '코끼리', '호랑이'], '조류', '어류']
print(animal)

for s in animal:
    print(s)
print()

bird = ['독수리', '까치', '참새']
fish = ['갈치', '붕어', '고등어']
animal[1:] = [bird, fish]
print(animal)

for lst in animal:
    for item in lst:
        print(item, end = ' ')
    print()
print()
```

```
[['사자', '코끼리', '호랑이'], '조류', '어류']
['사자', '코끼리', '호랑이']
조류
어류

[['사자', '코끼리', '호랑이'], ['독수리', '까치', '참새'], ['갈치', '붕어', '고등어']]
사자 코끼리 호랑이
독수리 까치 참새
갈치 붕어 고등어
```

동물을 리스트로 표현한 후 부분 리스트에 다른 리스트를 대입해 동물 이름 리스트를 포유류, 조류, 어류 순으로 얻을 수 있게 했다.

부분 대입을 통해 부분 리스트의 내용을 리스트로 대체할 수 있고, 중첩 for문으로 리스트 내부의 항목을 참조할 수 있다.

2.1 리스트의 부분 참조인 슬라이싱

리스트 역시 콜론을 사용해 리스트 전체나 일부분을 참조할 수 있다. 리스트[start : end : step]과 같이 사용한다. 첨자 start와 stop은 생략하면 마지막 항목까지 참조한다. 따라서 [:] 형식의 슬라이싱은 리스트 전체를 참조한다.

```
wlist = ['밥', '삶', '길', '죽', '꿈', '차', '떡', '복', '말']
print('wlist[:] = ', wlist[:])
print('wlist[:] = ', wlist[:])
print('wlist[::] = ', wlist[::])
print('wlist[::] = ', wlist[::-1])

#오름차순
print(wlist[:3])
print(wlist[1:3])
print(wlist[2:3])

#내림차순
print(wlist[::-2])
print(wlist[-1:-8:-3])

#오름과 내림차순의 혼재
print(wlist[1:-1:])
print(wlist[-2:-9:-3])
```

```
wlist[:] = ['밥', '삶', '길', '죽', '꿈', '차', '떡', '복', '말']
wlist[:] = ['밥', '삶', '길', '죽', '꿈', '차', '떡', '복', '말']
wlist[::] = ['말', '복', '떡', '차', '꿈', '죽', '길', '삶', '밥']
['밥', '죽', '떡']
['삶', '꿈', '복']
['길', '차', '말']
['말', '떡', '꿈', '길', '밥']
['말', '차', '길']
['삶', '길', '죽', '꿈', '차', '떡', '복']
['복', '꿈', '삶']
```

0에서 시작하는 첨자는 오름차순 정렬이다.

-1로 시작하는 역순 첨자는 내림차순이다.

정수 첨자와 음수 첨자를 함께 사용하는 슬라이싱 역시 가능하다.

2.2 리스트의 부분 수정

리스트의 일부를 다른 리스트로 수정하려면 슬라이스 방식에 대입해야 한다.

만일 슬라이스 방식으로 대입하지 않고 수정하면 항목으로 따로 대입된다. 이와 반대로 항목에 리스트를 대입하면 항목이 리스트로 대체된다.

리스트의 슬라이스에 동일한 리스트의 슬라이스를 대입하거나, 다른 리스트의 슬라이스를 대입해도 상관 없다.

```
>>> sports = ['풋살', '족구', '비치사커']
>>> sports[0:2] = ['축구']
>>> print(sports)
['축구', '비치사커']
```

2.3 리스트의 항목 삽입과 삭제

```
item = ['연필', '볼펜']
print(item)

#연필 1개와 볼펜 세 자루 삽입
item.insert(1, 2)
item.insert(3, 3)

#맨 뒤에 지우개, 1개 삽입
item.insert(4, '지우개')
item.insert(5, 1)
#현재 학용품 품목 출력
print(item)

#연필 두 자루 삭제
print(item.pop(1)) # 첨자 1 항목 삭제 삭제된 것을 반환하므로 삭제된 내용을 출력할수있음
item.remove('연필')
del item[2:] # 지우개와 수 품목 삭제

print(item)
```

리스트의 첨자 위치에 항목을 삽입하려면 리스트.insert(첨자, 항목)을 이용한다. 삭제하려면 메소드 remove(값) 또는 pop(첨자), pop()을 사용한다.

문장 del은 뒤에 위치한 변수나 항목을 삭제한다. 슬라이스로 리스트의 일부분을 삭제할 수도 있다.

clear()를 사용하면 모든 항목을 제거해 빈 리스트로 만든다.

2.4 리스트의 추가, 연결과 반복

리스트 메소드 `extend(list)`는 리스트에 인자인 `list`를 가장 뒤에 추가한다.

`+`는 리스트와 리스트를 연결한다. `*`를 이용해 리스트와 정수를 곱하면 지정된 정수만큼 반복된 리스트를 반환한다.

2.5 리스트 항목의 순서와 정렬

```
word = list('삶꿈정')
word.extend('복빛')
print(word)
word.sort() #오름차순 정렬
print(word)

fruit = ['복숭아', '자두', '골드키위', '귤']
print(fruit)
fruit.sort(reverse=True) #내림차순 정렬
print(fruit)

mix = word + fruit
print(sorted(mix))
print(sorted(mix, reverse=True))
```

리스트 메소드 `reverse()`는 항목 순서를 반대로 뒤집는다.

리스트 메소드 `sort()`는 리스트 항목의 순서를 오름차순으로 정렬한다. 내림차순으로 정렬하고 싶을 땐 `sort(reverse=True)`로 호출한다.

내장 함수 `sorted()`는 리스트의 항목 순서를 오름차순으로 정렬한 새로운 리스트를 반환한다. 원래의 리스트 자체는 변화되지 않는다는 점에 주의하자.

```
['삶', '꿈', '정', '복', '빛']
['꿈', '복', '빛', '삶', '정']
['복숭아', '자두', '골드키위', '귤']
['자두', '복숭아', '귤', '골드키위']
['골드키위', '귤', '꿈', '복', '복숭아', '빛', '삶', '자두', '정']
['정', '자두', '삶', '빛', '복숭아', '복', '꿈', '귤', '골드키위']
```

리스트 `word`는 함수 `extend()`를 이용해 복, 빛 인자가 가장 뒤에 추가됐다. 함수 `sort()`를 이용해 오름차순 정렬됐다.

리스트 `fruit`은 `sort(reverse=True)`를 이용해 내림차순 정렬됐다.

리스트 `mix`는 `word`와 `fruit`이 연결된 리스트이고, `sorted()`함수로 오름차순으로 정렬한 새로운 리스트가 반환되었다.

2.6 리스트 컴프리헨션

리스트 컴프리헨션은 리스트를 만드는 간결한 방법을 제공한다. 리스트를 만드므로 대괄호 []로 감싸고, 구성 항목인 *i*를 가장 앞에 배치하며, *for*문이 오는데 *range* 뒤에 콜론이 빠진다. 즉, 리스트 컴프리헨션을 사용하면 어떤 조건의 반환값을 항목으로 갖는 리스트를 쉽게 구성할 수 있다. 이는 함축, 축약, 내포, 내장 등으로도 불린다.

조건이 있는 컴프리헨션을 만들기 위해선 변수 *i*의 조건을 뒤에 붙이면 된다.

```
#for 문으로 리스트 생성
a = []
for i in range(10):
    a.append(i)
print(a)
```

```
#컴프리헨션으로 리스트 생성
seq = [i for i in range(10)]
print(seq)
```

```
#for 문으로 리스트 생성
s = []
for i in range(10):
    if i % 2 == 1:
        s.append(i ** 2)
print(s)
```

```
#컴프리헨션으로 리스트 생성
squares = [i ** 2 for i in range(10) if i % 2 == 1]
print(squares)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 9, 25, 49, 81]
[1, 9, 25, 49, 81]
```

```
리스트 = []
리스트 = [항목연산식 for 항목 in 시퀀스 if 조건식]
for 항목 in 시퀀스:
    if 조건식:
        리스트.append(항목연산식)
```

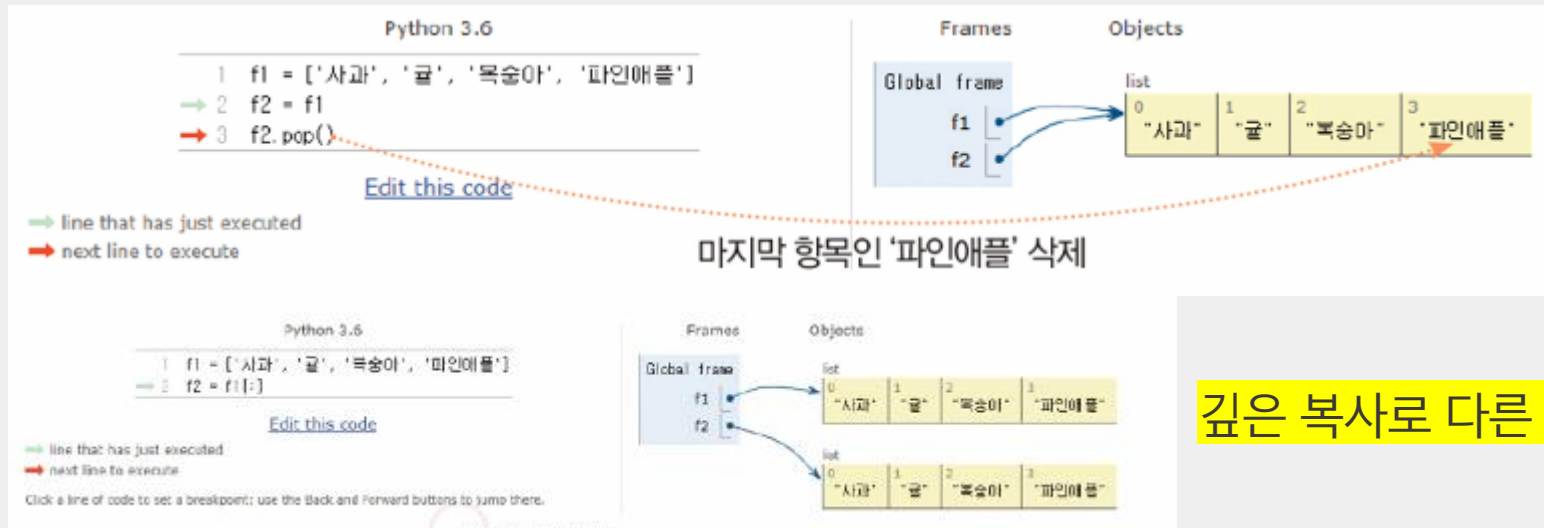
컴프리헨션 으로 리스트를 생성했을 때 *for* 문을 사용해 리스트를 생성하는 것보다 간단하게 생성되는 것을 볼 수 있다.

2.7 리스트 대입과 복사

리스트에서 대입 연산자 `=`는 얇은 복사라고 해서 대입되는 변수가 동일한 시퀀스를 가리킨다. 대입 연산자를 사용하면 동일한 메모리 공간을 사용하는 리스트를 공유하게 되므로, 한 변수에 `pop()` 함수 등을 사용해 변수를 삭제하면 두 변수의 리스트에서 그 항목이 함께 삭제된다.

완전히 새로운 리스트를 만들어 복사하려면 슬라이스 `[:]`나 `copy()` 또는 `list()` 함수를 사용해야 한다. 이 방법을 사용하면 항목이 같은 새로운 리스트가 된다. 이러한 복사를 깊은 복사라 한다.

변수 2개가 동일한 메모리를 공유하는지 검사하기 위해선 `print(f1 is f2)`와 같이 문장 `is`를 사용하면 된다.



얇은 복사로 동일한 리스트를 공유하는 모습

깊은 복사로 다른 리스트가 생성된 모습

3.1 괄호로 정의하는 시퀀스 튜플

튜플은 문자열, 리스트와 같은 항목의 나열인 시퀀스이다. 리스트와 달리 항목의 순서나 내용의 수정이 불가능하다.

콤마로 구분된 항목들의 리스트로 표현되며, 각각의 항목은 제한이 없다.

튜플은 ()로 만든다. 빈 튜플은 함수 tuple()로도 생성할 수 있다. 항목이 하나인 튜플을 표현할 때는 마지막에 콤마를 반드시 붙여야 하며, 붙이지 않으면 변수로 저장된다.

리스트와 같이 첨자 참조 tuple[index]와 tuple[start : stop : step]슬라이스가 가능하다. 그러나 수정은 할 수 없다.

```
singer = ('BTS', '불빨간사춘기', 'BTS', '블랙핑크', '태연')
song = ('작은 것들을 위한 시', '나만, 봄', '소우주', 'Kill This Love', '사계')
print(singer)
print(song)

print(singer.count('BTS'))
print(singer.index('불빨간사춘기'))
print(singer.index('BTS'))
print()

for _ in range(len(singer)):
    print("%s: %s" % (singer[_], song[_]))
```

수정이 아닌 참조에 해당하는 메소드 count()와 index()는 튜플에서 사용할 수 있다.

그러나 append(), insert(), pop()등과 같이 수정이 발생하는 메소드는 튜플에 없으므로 사용할 수 없다.

다음과 같이 for문으로 튜플의 모든 값을 출력할 수 있다. 특히 실행된 결괏값이 저장되는 변수 _를 사용하면 편리하다.

3.2 튜플 연결과 반복, 정렬과 삭제

리스트와 같이 +와 *는 각각 튜플을 연결하고 항목이 횟수만큼 반복된 튜플을 반환한다.

내장 함수 sorted()은 튜플 항목의 순서를 오름차순으로 정렬한 새로운 리스트를 반환한다.

함수 sorted(튜플, reverse=True)는 내림차순으로 정렬된 리스트를 반환한다. 절대 튜플 자체가 수정되지는 않는다.

문장 del에서 변수를 지정하면 변수 자체가 사라진다.

```
day1 = ('monday', 'tuesday', 'wednesday')
day2 = ('thursday', 'friday', 'saturday')

#('sunday')로 하면 튜플이 아니고 문자열

day3 = ('sunday', )

day = day1 + day2 + day3

print(type(day))
print(day)

day = day1 + day2 + day3 * 3
print(day)
```

```
<class 'tuple'>
('monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'saturday', 'sunday')
('monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'saturday', 'sunday', 'sunday', 'sunday')
```

+를 사용해 튜플을 연결하고, *를 이용해 정수를 곱해 세 번 반복하게 하였다.

```
>>> fruit = ('사과', '귤', '복숭아', '파인애플')
>>> tup = sorted(fruit)
>>> print(tup)
['귤', '복숭아', '사과', '파인애플']
>>> print(sorted(fruit, reverse=True))
['파인애플', '사과', '복숭아', '귤']
```

함수 sorted()를 이용한 튜플 정렬

Chapter 6

키와 값의 나열인 딕셔너리와 중복을 불허하는 집합



Section 01 키와 값인 쌍의 나열인 딕셔너리

Section 02 중복과 순서가 없는 집합

1.1 딕셔너리의 개념과 생성

딕셔너리는 키와 값의 쌍인 항목을 나열한 시퀀스이다. 콤마로 구분된 항목들의 리스트로 표현된다.

각각의 항목은 키:값과 같이 키와 값을 콜론으로 구분하고 전체는 중괄호 {}로 묶는다.

딕셔너리 역시 빈 중괄호로 빈 딕셔너리를 만들 수 있다. 내장 함수 dict() 호출로도 빈 딕셔너리를 생성할 수 있다.

빈 딕셔너리에 항목을 넣으려면 딕셔너리[키] = 값의 문장을 사용해야 한다. 항목 값으로 리스트나 튜플 등도 가능하다.

```
lect = dict()
lect['강좌명'] = '파이썬 기초'
lect['개설년도'] = [2020, 1]
lect['학점시수'] = (3, 3)
lect['교수'] = '김민국'
print(lect)
print(len(lect))
print()

print(lect['개설년도'], lect['학점시수'])
print(lect['강좌명'], lect['교수'])
```

```
{'강좌명': '파이썬 기초', '개설년도': [2020, 1], '학점시수': (3, 3), '교수': '김민국'}
4

[2020, 1] (3, 3)
파이썬 기초 김민국
```

딕셔너리 값은 대괄호를 사용한 딕셔너리[키]로 해당 값을 참조할 수 있다.

함수 print()로 딕셔너리를 출력하면 중괄호의 딕셔너리 형태로 출력할 수 있다.

딕셔너리는 중괄호 사이에 키와 값의 항목을 기술한다.

항목 순서는 의미가 없으며, 키는 중복될 수 없다.

키는 수정될 수 없지만 값은 수정될 수 있다.

값은 키로 참조된다.

1.2 다양한 인자의 함수 dict()로 생성하는 딕셔너리

인자로 리스트나 튜플을 사용할 수 있다. 그 내부에서 [키, 값] 리스트 형식과 (키, 값) 튜플 형식을 모두 사용할 수 있다. 키가 단순 문자열이면 간단히 키=값 항목 나열로도 지정할 수 있다.

```
>>> day = dict([['월', 'monday'], ('화', 'tuesday'), ['수', 'wednesday']])
>>> print(day)
{'월': 'monday', '화': 'tuesday', '수': 'wednesday'}
```

```
>>> day = dict(월='monday', 화='tuesday', 수='wednesday')
>>> print(day)
{'월': 'monday', '화': 'tuesday', '수': 'wednesday'}
```

1.3 딕셔너리 키는 수정 불가능한 객체로 사용

딕셔너리의 키는 수정 불가능한 객체는 모두 사용 가능하다. 따라서 정수, 실수 등이 모두 가능하다. 수정 불가능한 튜플은 가능하지만, 리스트는 수정할 수 있으므로 불가능하다. 딕셔너리에서 대괄호 안의 객체는 키이므로, 첨자처럼 참고할 수 없다.

```
>>> month = { 1: 'January', 2: 'February', 3.0: 'March' }
>>> print(month)
{1: 'January', 2: 'February', 3.0: 'March'}
>>> print(month[2])
February
```

1.4 딕셔너리 항목의 순회

딕셔너리 메소드 `keys()`는 키로만 구성된 리스트를 반환한다.

`for`문을 사용해 딕셔너리의 모든 항목을 참조하는 구문을 만들 수 있다.

메소드 `items()`는 (키, 값) 쌍의 튜플이 들어 있는 리스트를 반환한다.

`for`문을 사용해 딕셔너리의 모든 항목을 참조하는 구문을 만들 수 있다.

메소드 `values()`는 값으로만 구성된 리스트를 반환한다.

반복 `for`문에서는 딕셔너리 변수만으로 모든 키를 순회할 수 있다.

```
>>> day = dict(월='monday', 화 = 'tuesday', 수 = 'wednesday')
>>> for key in day.keys():
    print('%s요일 %s' % (key, day[key]))
```

```
월요일 monday
화요일 tuesday
수요일 wednesday
```

```
>>> for key, value in day.items():
    print('%s요일 %s' % (key, value))
```

```
월요일 monday
화요일 tuesday
수요일 wednesday
```

```
>>> day = dict(월='monday', 화 = 'tuesday', 수 = 'wednesday')
>>> print(day.values())
dict_values(['monday', 'tuesday', 'wednesday'])
```

```
>>> day = dict(월='monday', 화 = 'tuesday', 수 = 'wednesday')
>>> for key in day:
    print('%s요일 %s' % (key, day[key]))
```

```
월요일 monday
화요일 tuesday
수요일 wednesday
```

1.5 딕셔너리 항목의 참조와 삭제

딕셔너리 메소드 `get(키)`는 키의 해당 값을 반환한다. 딕셔너리에 키가 없어도 `None`을 반환한다. 만일 키 뒤에 다른 인자를 넣으면 키가 없을 때 이 저장된 값을 반환한다.

메소드 `pop()`은 키인 항목을 삭제하고, 삭제되는 키의 해당 값을 반환한다.

메소드 `popitem()`은 임의의 (키, 값) 튜플을 반환하고 삭제한다. 데이터가 없다면 오류가 발생한다.

문장 `del`에 이어 키로 지정하면 해당 항목이 삭제된다.

1.6 딕셔너리 항목 전체 삭제와 변수 제거

딕셔너리 메소드 `clear()`는 기존의 모든 키 : 값 항목을 삭제한다.

문장 `del()`에 이어 키로 지정하면 변수 자체가 메모리에서 제거된다.

1.7 딕셔너리 결합과 키의 멤버십 검사 연산자 `in`

메소드 `update()`는 인자인 다른 딕셔너리를 합병한다. 원 딕셔너리와 동일한 키가 있다면 인자 딕셔너리의 값으로 대체된다.

문장 `in`으로 딕셔너리에 키가 존재하는지 간단히 검사할 수 있다. 값으로 조회하면 항상 `False`이며, `not in`으로도 검사 가능하다.

2.1 수학에서 배운 집합을 처리하는 자료형

파이썬에서 집합은 수학과 같이 원소를 콤마로 구분하여 중괄호로 둘러싸 표현한다.

원소는 불변 값으로 중복될 수 없으며 서로 다른 값이어야 한다. 원소의 순서는 의미가 없다.

집합의 원소는 수정 불가능한 것이어야 하며, 리스트와 같은 가변적인 것은 허용되지 않는다.

2.2 내장 함수 set()을 활용한 집합 생성

집합은 내장 함수 set()으로도 생성할 수 있다. set(원소로 구성된 리스트_ or 튜플_ or 문자열_)

인자가 없으면 공집합이 생성되며, 인자는 반복적이면 가능하다. 그러나 항목은 불변적이어야 한다. 수정될 수 있는 리스트나 딕셔너리는 허용되지 않는다.

함수 set()에서는 인자로 리스트 또는 튜플 자체를 사용할 수 있으며, 결과는 항목에서 중복을 제거한 원소이다.

인자로 문자열이 사용되면 각각의 문자가 원소인 집합이 생성된다.

```
>>> s = set()
>>> type(s)
<class 'set'>
>>> s
set()
>>> set([1, 2, 2, 3])
{1, 2, 3}
>>> set('abc')
{'a', 'c', 'b'}
```

Section 02 중복과 순서가 없는 집합

2.3 중괄호로 직접 원소를 나열해 집합 생성

중괄호 안에 직접 원소를 콤마로 구분하는 방법으로 집합을 생성할 수 있다. 원소는 변할 수 없는 것이어야 한다.

```
planets = set('해달별')
fruits = set(['감', '귤'])
nuts = {'밤', '잣'}
things = {('밤', '잣'), ('감', '귤'), '해달'}
#things = {['밤', '잣'], ('감', '귤'), '해달'} # 오류 발생

print(planets)
print(fruits)
print(nuts)
print(things)
```

```
{'달', '해', '별'}
{'귤', '감'}
{'잣', '밤'}
{('밤', '잣'), ('감', '귤'), '해달'}
```

2.4 집합의 원소 추가와 삭제

이미 만들어진 집합에 원소를 추가할 때는 메소드 `add()`를 사용한다.

원소의 삭제는 메소드 `remove()`, `discard()`, `pop()`을 사용한다.

메소드 `discard()`를 사용하면 원소가 없어도 오류가 발생하지 않고, `pop()`을 사용하면 임의의 원소를 삭제한다.

집합의 모든 원소를 삭제하려면 메소드 `clear()`를 사용한다.

2.5 집합의 주요 연산인 합집합, 교집합, 차집합, 여집합

합집합, 교집합, 차집합, 여집합 등을 쉽게 구할 수 있다.

합집합은 연산자 `|` 혹은 메소드 `union()`을 사용한다. 이들은 합집합을 반환하며 값이 수정되지는 않는다..

메소드 `a.update(b)`도 같은 효과가 있지만, 합집합 결과가 `a`에 반영돼 수정되는 차이점이 있다. 대입연산자 `|=`의 결과와 같다.

교집합은 연산자 `&`와 메소드 `intersection()`을 사용한다. 집합 `a`, `b` 모두에 영향을 미치지 않는다.

`a.intersection_update(b)`로 사용하면 집합 `a`를 교집합으로 수정한다. 대입연산자는 `&=`의 결과와 같다.

차집합은 연산자 `-`와 메소드 `difference()`를 사용한다. 교환 법칙이 성립하지 않는다. 대입연산자는 `-=`이다.

여집합은 대칭 차집합이라고도 부르며, 연산자 `^`와 메소드 `symmetric_difference()`를 사용한다. 대입연산자는 `^=`이다.

```
daysA = {'월', '화', '수', '목'}
daysB = set(['수', '목', '금', '토', '일'])
weekends = set(['토', '일'])

alldays = daysA | daysB
print(alldays)

workdays = alldays - weekends
print(workdays)

print(daysA & daysB)
print(daysA.symmetric_difference(daysB))
```

```
{'금', '토', '화', '목', '월', '일', '수'}
{'금', '화', '목', '월', '수'}
{'수', '목'}
{'금', '토', '월', '일', '화'}
```

2.6 함수 len()과 소속 연산 in

함수 len()으로 집합에 들어 있는 원소의 개수를 확인할 수 있다.

```
>>> p1 = {'fortran', 'basic', 'cobol', 'c'}
>>> len(p1)
4
```

소속 연산자 in은 집합에서 유일하게 사용될 수 있다. 특정 원소가 집합에 있는지 확인하기 위해 사용된다.

```
>>> 'fortran' in p1
True
>>> 'python' in p1
False
```

소감



소감

초보자가 배우기에 좋은 쉽고 간결한 언어라는 평판과는 다르게, 사실 파이썬은 자바나 C보다 저에게 낯선 언어였습니다. 1학년때 배운 언어가 HTML이나 C, 자바 위주이기 때문이기도 했지만, 저에게는 왠지 어려운 단어라는 느낌이 들었기 때문입니다. 자주 접했던 단어와는 다른 점이 많은 문법과 (for문을 쓰는 방법 등) 왠지 어려워 보이는 이름이 제 그런 인식에 한 몫을 하지 않았나 싶습니다.

하지만 직접 배워보니 왜 지금까지 어렵다고 생각해서 손도 대지 않았을까 싶을 정도로 강력한 기능이 많다는 것을 직접 체감할 수 있게 되었습니다. 아직 배워가는 단계이긴 하지만, 책에서 본 내용과 강의들의 내용을 토대로 생각해 보면 왜 최근에 가장 각광받고 있는 프로그래밍 언어인지 알 수 있었습니다.

중간고사로 포트폴리오를 작성하며 평소에는 집중하지 않았던 책의 부분도 집중해서 보다 보니, 알고 있던 부분은 더 정확하게 알 수 있게 되었고 모르던 부분을 체득할 수 있게 되었습니다. 앞으로도 파이썬 강의를 열심히 들어 이 강력한 언어를 원하는 대로 사용할 수 있게 되었으면 좋겠습니다.

늘 양질의 강의를 들려 주시는 교수님께 감사하다는 말로 끝을 맺겠습니다. 감사합니다!
